

Neither Quick Nor Proper – Evaluation of QuickProp for Learning Deep Neural Networks

Clemens-Alexander Brust, Sven Sickert, Marcel Simon, Erik Rodner, and
Joachim Denzler

Computer Vision Group, Friedrich Schiller University Jena, Germany
<http://www.inf-cv.uni-jena.de>

Abstract. Neural networks and especially convolutional neural networks are of great interest in current computer vision research. However, many techniques, extensions, and modifications have been published in the past, which are not yet used by current approaches. In this paper, we study the application of a method called QuickProp for training of deep neural networks. In particular, we apply QuickProp during learning and testing of fully convolutional networks for the task of semantic segmentation. We compare QuickProp empirically with gradient descent, which is the current standard method. Experiments suggest that QuickProp can not compete with standard gradient descent techniques for complex computer vision tasks like semantic segmentation.

1 Introduction

Convolutional neural networks (CNNs) [10] achieve state-of-the-art performance in many areas of computer vision including image classification, object detection, and segmentation [6,9,11,15]. However, the training time of large networks is a major bottleneck when evaluating different architectures and new types of layers. In this work, we study the QuickProp algorithm [4] for decreasing training time in the domain of semantic segmentation and potentially reaching a better optimum during training.

QuickProp [4] is a second-order optimization algorithm that uses a simple approximation of the Hessian’s diagonal to accelerate optimization and therefore belongs to the class of Quasi-Newton algorithms. So far, it has been evaluated and studied only for standard neural network training [12,17,18]. However, current neural network architectures, like CNNs, are characterized by a significantly larger number of parameters. Furthermore, the increased number of layers leads to a higher numerical error when computing the gradients with back-propagation [7]. For an evaluation of its performance, we test the algorithm in synthetic and real-world experiments and compare its behavior to the traditional and widely used optimization approach gradient descent (GD).

In this work, we focus on the task of semantic segmentation. Each pixel of an image is classified into known classes and thereby the image is segmented into meaningful regions. In contrast to unsupervised segmentation, this not only

returns possible object boundaries as regions but also semantic labels for each of those regions. For this task, we make use of the convolutional network architectures as proposed in [1] and use them throughout all our experiments.

The remainder of the paper is structured as follows. In Section 2, we give a brief review of related work and optimization techniques for the task of network training. In Section 3, we elaborate on the QuickProp algorithm which was originally introduced in [4]. An evaluation using synthetic and real-world experiments is given in Section 4. A discussion and summary in Section 5 concludes the paper.

2 Related work

The most related approach to QuickProp is Newton optimization, where the Hessian of the objective function is used for a second-order approximation. However, computing the Hessian exactly as done in the Newton method is infeasible for neural networks with more than a few thousand parameters. For example, current state-of-the-art architectures in the computer vision area usually parameters in the order of millions [9]. Therefore, Quasi-Newton methods are required approximating the Hessian in a computationally efficient manner. In fact, QuickProp uses a simple approximation that comes with nearly no additional computational cost. Other alternative Quasi-Newton methods are reviewed in [13].

To speed up QuickProp even further, the work of [2] magnifies the gradient of sigmoid activation layers inspired by the technique proposed by [14]. This method has no effect on the learning with state-of-the-art CNN architectures, since rectified linear units are used, whose gradients are either zero or one and can therefore not be exponentially magnified. Another modification is given by [17,18], where absolute differences of the gradients and former weight changes are used and multiplied with a tuning parameter η . The authors also show the convergence of QuickProp for continuously differentiable objectives satisfying several additional conditions also for the gradients. A comparison of several optimization algorithms including QuickProp is given by [12], but limited to standard neural networks with a relatively small number of parameters.

A very common optimization strategy used for training deep neural networks is gradient descent with momentum as studied in [16]. Similar to QuickProp, momentum incorporates the change of weights of the previous iteration. However, this is done in an additive manner, whereas QuickProp performs a multiplicative update and also handles each component of the weight vector independently. Other methods including AdaDelta [19], AdaGrad [3] and Adam [8] try to improve over gradient descent by estimating a optimal learning rate in each update step. In contrast, QuickProp assumes the output to be a quadratic function of each variable and calculates the location of the optimum directly.

3 QuickProp optimization algorithm

Quick Propagation or QuickProp [4] is an iterative second-order optimization algorithm. The algorithm approximates in each step the objective function with a quadratic function for each variable independently from the others. In the case of neural networks, the input variables of the objective functions are the parameters or weights of the network.

QuickProp optimization step Let $E(\mathbf{w})$ be the error (or loss) function of a neural network with respect to the parameters $\mathbf{w} = (w_1, w_2, \dots, w_K)$. The goal of training is to minimize $E(\mathbf{w})$ for the training set.

In each iteration, QuickProp determines the new weight $\mathbf{w}^{(t+1)}$ given the current weight $\mathbf{w}^{(t)}$ as well as the slope $\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}}$ of the previous and current step. Gradient descent updates the weights by calculating $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta \mathbf{w}^{(t-1)} = \mathbf{w}^{(t)} + \gamma \cdot \mathbf{g}^{(t)}$ with $\gamma \in \mathbb{R}$ being the learning rate or step length and $\mathbf{g}^{(t)} = \nabla_{\mathbf{w}} E(\mathbf{w}^{(t)})$ being the gradient of E evaluated at $\mathbf{w}^{(t)}$. In QuickProp, each weight w_i is optimized independently and given a selected component i , all weights except $w_i^{(t)}$ are assumed to be fixed. We therefore assume \mathbf{w} to be a scalar w in the following. To derive an update rule for $w^{(t+1)}$, E is approximated by a parabola using a second-order Taylor approximation assuming that E is twice differentiable:

$$E(w^{(t+1)}) \approx E(w^{(t)}) + g^{(t)} \cdot \Delta w^{(t)} + \frac{1}{2} E''(w^{(t)}) (\Delta w^{(t)})^2 \quad (1)$$

Whereas this is a standard approximation used by all (Quasi-)Newton methods, QuickProp now approximates the second derivative E'' by a finite difference using $w^{(t)}$ and $w^{(t-1)}$:

$$E''(w^{(t)}) \approx \frac{g^{(t)} - g^{(t-1)}}{\Delta w^{(t-1)}} \quad (2)$$

In contrast to the Hessian-free optimization technique presented in [13], this does not require an additional pass through the network for forward differentiation. However, it savings in computation time come with greater numerical instabilities that need to be tackled and are further discussed in the next section. By combining both equations and computing the stationary point of the approximation of E with respect to $\Delta w^{(t)}$, we get:

$$\Delta w^{(t)} = \frac{g^{(t)}}{g^{(t-1)} - g^{(t)}} \cdot \Delta w^{(t-1)} \quad (3)$$

which is the update step used by QuickProp. The local parabola approximation p of QuickProp around $w^{(t)}$ can be therefore expressed by:

$$p(z) = a \cdot (z - w^{(t)})^2 + b \cdot (z - w^{(t)}) + c, \text{ where} \quad (4)$$

$$a = \frac{1}{2} \frac{g^{(t)} - g^{(t-1)}}{\Delta w^{(t-1)}} \quad (5)$$

$$b = g^{(t)} \quad (6)$$

$$c = E(w^{(t)}) \quad (7)$$

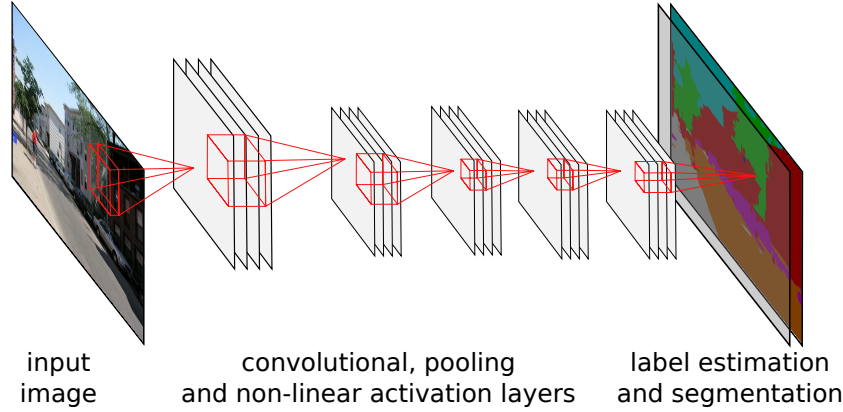


Fig. 1. Architecture of convolutional neural networks used for semantic segmentation.

Special cases Instead of adding $\Delta w^{(t)}$ immediately to the weights, the Quick-Prop algorithm handles the following three cases separately:

1. **The absolute value of the current slope is smaller than the previous one, but has the same sign:** In this case, the new weight is obtained by adding the delta, *i.e.* $w^{(t+1)} = w^{(t)} + \Delta w^{(t)}$. The step $\Delta w^{(t)}$ depends on the difference between the current and previous slope as described in eq. 3.
2. **The signs of the slopes differ:** The minimum lies between the current and the previous weight $w_i^{(t-1)}$ and $w_i^{(t)}$. The update rule from eq. 3 results in a backwards step because of the different signs.
3. **The absolute value of the current slope is larger or equal to the previous slope:** This would result in an infinite step or a step in the wrong direction. The step size needs to be limited. The author introduces the hyperparameter μ , called the “maximum growth factor”. A weight step can never be larger than μ times the previous step. Steps that are too large or infinite or in the wrong direction are replaced by μ times the previous step.

To initialize the algorithm, a traditional gradient descent step is taken. Gradient descent is also used for weights that did not change in an iteration, which would result in $\Delta w^{(t)}$ being 0 for all following time steps.

Implementation details Our implementation follows exactly the work of Fahlman et al. [4]. To ignite the optimization process or when the gradient becomes too small, we use a single fixed learning rate gradient descent step. We set this threshold to $1e-15$. The implementation also uses the maximum growth factor and adds a gradient step if the current and the previous gradient have the same signs.

4 Experiments

In order to evaluate the QuickProp algorithm in a modern context we use two datasets of different complexities for the task of semantic segmentation. In this section, we investigate how QuickProp compares to gradient descent which is the standard technique for training neural networks. After an introduction of the datasets and a description of the experimental setup, we analyze the influence of the network complexity.

4.1 Deep neural networks for semantic segmentation

Our experiments focus on the application of deep neural networks for semantic segmentation using so-called fully-convolutional neural networks [11]. The goal of semantic segmentation is to classify each pixel of a given image into one of K semantic categories. Fig. 1 shows the basic principle of the architecture of a CNN used for this task. The input of the network is a given image and the output is a score for each class and pixel. The index of the maximum score then corresponds to the semantic class assigned to the pixel.

The network is comprised of different layers, mainly (a) convolutional layers, (b) pooling layers, (c) up-sampling layers, and (d) non-linear activation layers. Since we do not focus on the application of semantic segmentation in our paper but rather on efficient optimization techniques for learning, we refer the interested reader to [1,11] for more details.

4.2 Experimental setup

In order to evaluate the performance of the optimization techniques for semantic segmentation tasks, we report typical quantitative measures in this field of research. With respect to the semantic labeling of pixels we follow [1] in measuring the performance using the overall accuracy and mean class-wise accuracy. Furthermore, we analyze the different optimization methods using the output of the quadratic loss function. More specifically, the reported *loss* in our experiments is the per pixel error averaged over all pixels of all examples and weighting updates. Additionally, all results shown in this section are averaged over several runs allowing for a more robust evaluation of the used techniques.

4.3 Datasets

For our experimental evaluation we used the following datasets which have different requirements on the type and complexity of the network architecture.

T.O.Y. dataset In order to analyze the performance for varying settings we created a small artificial dataset serving as a working example. This so-called T.O.Y. dataset consists of only a single image with two foreground classes and one background class. It is designed to be easy to solve even for a tiny network. The training image and its corresponding labeling is depicted in Fig. 2.



Fig. 2. Datasets used in our experiments: A simple three class scenario in the T.O.Y. dataset (left) and a multi-class setting of urban scenes in the LabelMeFacade dataset [5] (right). Depicted are both input image and the corresponding ground truth labeling as color coding.

The T.O.Y. dataset allows for the verification whether QuickProp yields reasonable solutions because the classes are differentiable by basic filter masks. A second advantage of this dataset is the small size. There is a trend to publish the results of just a single random initialization of a CNN, especially in competitive settings. However, analyzing the distribution over many repetitions provides results with statistical significance.

LabelMeFacade The second dataset we use in our experimental evaluation is LabelMeFacade [5] which consists of 945 images taken from urban scenes. It contains ground truth labels for eight classes: *building*, *road*, *pavement*, *sky*, *vegetation*, *window* and *door*. Additionally, there exists a background class which can be used to exclude undefined areas from the evaluation. Fig. 2 shows an example containing all labeled classes with their corresponding color coding.

In our evaluation we follow [5] and split the dataset into 845 images for testing and 100 images for training. For model selection and tuning of hyperparameters, we split the 100 training images again into 50 images each for training and 50 images for validation.

4.4 Comparison of QuickProp and gradient descent

T.O.Y. dataset We train a network 200 times from scratch with different random initializations. The training is stopped after 30 “epochs” of 10.000 iterations each and one image per iteration. The CNN architecture is a simple 2-layer CNN. One convolutional layer with two filter masks are followed by output neurons and hence corresponds to a simple template matching approach. The input to the network is exactly as large as the filters in the first layer. A quadratic loss is used in order to match the assumptions of QuickProp as close as possible.

In Fig. 3 we show how the loss is decreasing during training and subsequent testing of the networks after each epoch. It can be seen that in training QuickProp is outperformed by gradient descent. The results during testing, however, are inverted with respect to the reported loss. This behavior is interesting, since it suggests a better generalization ability of the network which was trained using

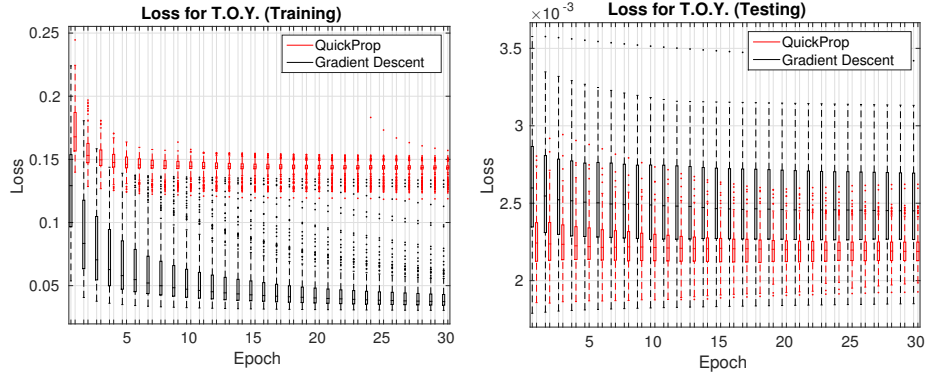


Fig. 3. Quantitative analysis using the T.O.Y. dataset: during training the loss using QuickProp is substantially worse than with gradient descent (left). However, during testing the relationship is inverted (right).

QuickProp. However, the visual variety of the T.O.Y. dataset is very small and hence a general statement is not possible.

The overall accuracy of the learned networks during testing was comparable. It saturated at 78% for both the gradient descent and the QuickProp approach after a few epochs. In contrast, the mean class-wise accuracy for QuickProp was significantly worse compared to the standard learning approach. While the gradient descent technique saturated at 62% after five epochs, QuickProp never rose above 41%.

LabelMeFacade dataset The architecture used for this set of experiments consists of three convolutional layers and two fully connected layers. Since we are using a fully convolutional setting the fully connected layers are also convolutional layers of kernel size 1×1 [11]. Each convolutional layer is followed by subsequent non-linearity layer. While the last non-linear layer is **Sigmoid**, the remaining activation functions are **Tanh**. We trained CNNs for ten epochs for each optimization method. The training process is repeated ten times.

Fig. 4 shows the loss over time during training as well as the accuracy during training and testing. As can be seen, QuickProp performs worse than gradient descent in this scenario. For each training epoch the error rates of Quickprop are higher and the resulting accuracy is up to five percent points below gradient descent. The reported accuracy for QuickProp during testing is also worse. Additionally, an increase in variance of the results can be observed.

In comparison to earlier experiments, the results for testing do not support the findings of the T.O.Y. experiments with respect to the generalization ability. However, it is worth noting that the LabelMeFacade dataset is in general more complex than the T.O.Y. dataset. While the latter is a simple binary classification task the former features a multi-class setting with a high variety of visual representations for the classes. As a consequence, the network in use was of higher complexity than the one for the T.O.Y. dataset. In order to investigate

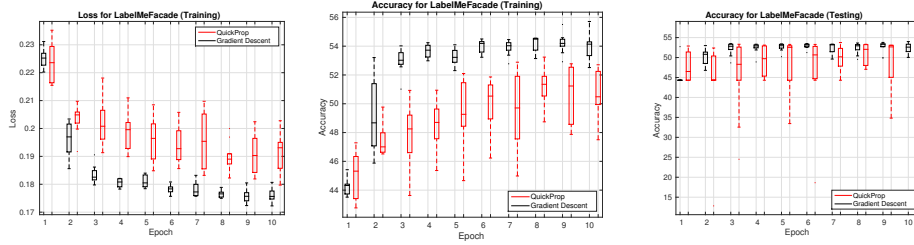


Fig. 4. Quantitative analysis using LabelMeFacade: During training and testing gradient descent achieves better results than QuickProp.

whether the architecture of the CNN plays a defining role in the performance of QuickProp we ran a second series of experiments on LabelMeFacade.

4.5 Network complexity analysis

The models studied so far range from a very simple network of only 109 parameters for the T.O.Y. dataset up to networks featuring more than 100.000 weights for LabelMeFacade. In this set of experiments we evaluate if the model complexity relates to the performance of the different training schemes. We propose two scalable network architectures based on the initial model we used for the LabelMeFacade experiments. The first analysis features a variable count of filters in a specific layer. After that, the network is modified in a way to enable a variable amount of layers.

Scaling the amount of filters First, we vary the number of filters in the second convolutional and the first fully connected layer. Both layers are linearly dependent on a variable k , using k filters for convolution and $12 \cdot k$ neurons in the fully connected layer. Accordingly, the total amount of parameters $|\mathbf{w}|$ in the networks can then be expressed in terms of k as $|\mathbf{w}| = 2473 + 4497 \cdot k$. In order to evaluate the influence of differently sized networks we train and test CNNs on the LabelMeFacade dataset for $k \in \{2, 7, 12, \dots, 22\}$.

In Fig. 5 we report the loss reached for networks of increasing size. As before, we compare networks trained using gradient descent and QuickProp. For each k the CNNs were learned for 100 epochs. Since the training took a very long time, we were only able to repeat the training once for each setup.

As can be seen from the results in Fig. 5, the loss during training for QuickProp is higher than using GD for almost all cases. Interestingly, the loss for the case $k = 2$ is the same for all runs. However, a general increase of the gap between both methods is visible for more complex networks. This supports the findings that QuickProp performs better for smaller networks in this application.

Scaling the amount of layers Another way of increasing the complexity of CNNs is to add more layers to its architecture. In order to achieve a linear growth in weights, we suggest to repeat the first fully connected layer with 192 kernels l times. As this layer is in fact a convolutional layer of size 1×1 , the

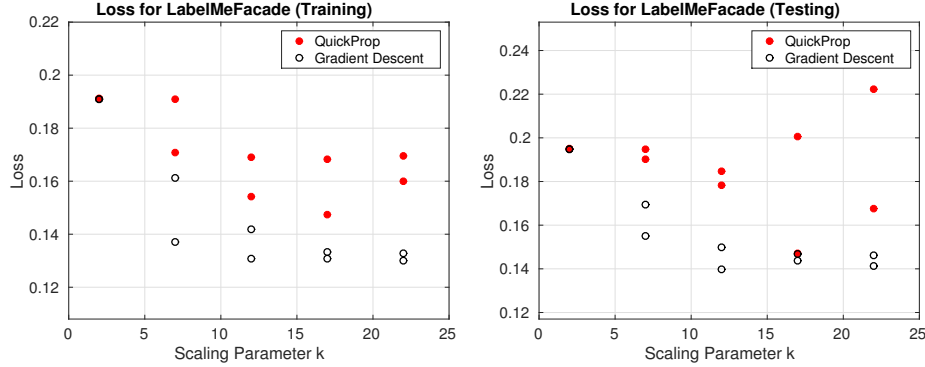


Fig. 5. Scaling the amount of filters per layer to increase network complexity:

total amount of parameters in the networks computes as $|\mathbf{w}| = 56437 + l$, with $l \in \{0, 1, \dots, 5\}$ being a linear scaling parameter. Experiments were conducted for a step-wise increased l and we compared the same optimization methods as before. We repeated each experimental setup once and report both runs in Fig. 6.

Similar to our earlier experiments, QuickProp performs worse when compared to the standard technique gradient descent. However, in comparison to the scaling of the amount of filters, the gap between the performance of both methods is not increasing. On the contrary, results during testing show a decreasing gap. However, it is worth noting that a simple repetition of the first fully connected layer seems to be unreasonable. For an increasing l the loss is getting worse and for $l > 2$ stays constant in our series of experiments. Also note, that without losing performance the first fully connected layer can be skipped altogether ($l = 0$). This might be due to the fully convolutional character of our network.

5 Conclusions

In this technical report, we evaluated QuickProp as an iterative second-order optimization algorithm for the training of convolutional neural networks. We conducted experiments for the task of semantic segmentation using a fully convolutional network configuration.

For an initial evaluation, a simple toy example as well was used allowing a high amount of repetitions of the experiments in order to retrieve results of statistical significance. While the QuickProp algorithm showed a higher loss during training, the performance during testing was superior to a network trained using gradient descent.

However, for a *real-world* scenario using a dataset consisting of urban street scenes, the standard approach of gradient descent still performs significantly better. In order to investigate a possible relationship between the size and complexity of networks to be trained and the performance of optimization techniques,

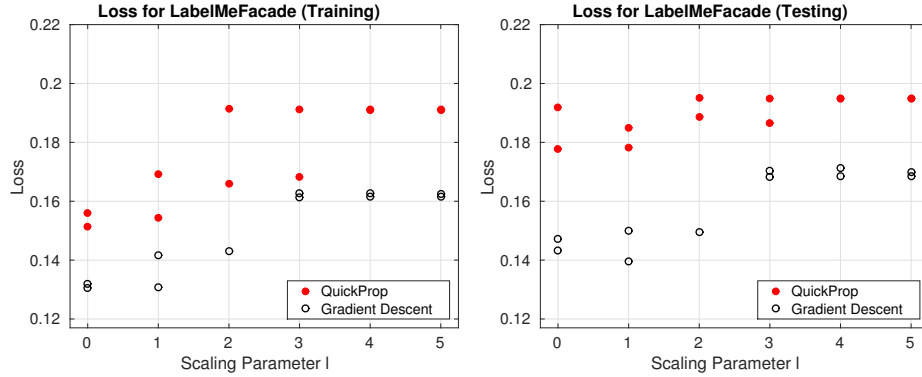


Fig. 6. Scaling the amount of layers to increase network complexity:

we ran scaling experiments of two different types. The series of experiments for a scalable amount of filter kernels in selected layers showed a degrading performance of QuickProp for larger networks when compared to gradient descent. In general, improved second-order optimization approaches, like QuickProp, can help in speeding up the training process and finding a better optimum. However, these cases seem to be rare and do not include practical scenarios featuring convolutional neural networks. Since the models of recently proposed networks for computer vision task become larger, we can not recommend to use QuickProp without modifications in contemporary architectures and tasks.

References

1. Clemens-Alexander Brust, Sven Sickert, Marcel Simon, Erik Rodner, and Joachim Denzler. Convolutional patch networks with spatial prior for road detection and urban scene understanding. In *Proceedings of the International Conference on Computer Vision Theory and Applications*, pages 510–517, 2015.
2. Chi-Chung Cheung, Sin-Chun Ng, and Andrew K Lui. Improving the quickprop algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–6. IEEE, 2012.
3. John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.
4. Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. *Neural Networks*, 6(3):1–19, 1988.
5. Björn Fröhlich, Erik Rodner, and Joachim Denzler. Semantic segmentation with millions of features: Integrating multiple cues in a combined random forest approach. In *Proceeding of the Asian Conference on Computer Vision*, pages 218–231, 2012.
6. Kaiming He, Xiangyu Zhang, Shaoqing, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2016.
7. Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: The difficulty of learning long-term dependencies, 2001.

8. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
9. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.
10. Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
11. Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, Boston, MA, USA, June 2015.
12. George D. Magoulas, Michael N. Vrahatis, and George S. Androulakis. Improving the convergence of the backpropagation algorithm using learning rate adaptation methods. *Neural Computation*, 11(7):1769–1796, 1999.
13. James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. Springer, 2012.
14. Sin-Chun Ng, Chi-Chung Cheung, and Shu-Hung Leung. Magnified gradient function with deterministic weight modification in adaptive learning. *IEEE Transactions on Neural Networks*, 15(6):1411–1423, 2004.
15. Marcel Simon and Erik Rodner. Neural activation constellations: Unsupervised part model discovery with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.
16. Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning*, pages 1139–1147, 2013.
17. Michael N. Vrahatis, George D. Magoulas, and Vassilis P. Plagianakos. Convergence analysis of the quickprop method. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1209–1214. IEEE, 1999.
18. Michael N. Vrahatis, George D. Magoulas, and Vassilis P. Plagianakos. Globally convergent modification of the quickprop method. *Neural Processing Letters*, 12(2):159–170, 2000.
19. Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.